FindResearch.org: How to Encourage Sharing of Research Artifacts

Christian Collberg Todd Proebsting

Department of Computer Science
University of Arizona

http://repeatability.cs.arizona.edu

http://findresearch.org

Communications of the ACM, March 2016

Supported by *the private foundation that must not be named*

Opening Gambit
The Deception Study
Sharing Proposals
How to Share?
Pushback

Some Computer Security Paper

Really Good Computer Science Department
Really Good School

Abstract

We present a new general technique for protecting clients in distributed systems against *Remote Man-at-the-end* (R-MATE) attacks. Such attacks occur in settings where an adversary has physical access to an untrusted client device and can obtain an advantage from tampering with the hardware itself or the software it contains.

In our system, the trusted server overwhelms the untrusted client's analytical abilities by continuously and automatically generating and pushing to him diverse client code variants. The diversity subsystem employs a set of primitive code transformations that provide an ever-changing attack target for the adversary, making tampering difficult without this being detected by the server.

1. Introduction

Man-at-the-end (MATE) attacks occur in settings where an adversary has physical access to a device and compromises it by tampering with its hardware or software. *Remote* man-at-the-end (R-MATE) attacks occur in distributed systems where *untrusted clients* are in frequent communication with *trusted servers* over a network, and malicious user can get an advantage by compromising an untrusted device.

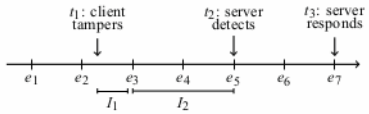
To illustrate the ubiquity of R-MATE vulnerabilities, consider the following four scenarios. First, in the *Advanced Metering Infrastructure* (AMI) for controlling the electrical power grid, networked devices ("*smart meters*") are installed at individual house-holds to allow two-way communication with control servers of the utility company. In an R-MATE attack against the AMI, a malicious consumer tampers with the meter to emulate an imminent blackout, or to trick a control server to send disconnect commands to other customers [7, 21]. Second, massive multiplayer online games are susceptible to R-MATE attacks since a malicious player who tampers with the game client can get an advantage over other players [16]. Third, wireless sensors are often deployed in unsecured environments (such as theaters of war) where they are vulnerable to tampering attempts. A compromised sensor could be coached into supplying the wrong observations to a base station, causing real-world damage. Finally, while electronic health records (EHR) are typically protected by encryption while stored in databases and in transit to doctors' offices, they are vulnerable to R-MATE attack if an individual doctor's client machine is compromised.

1.1 Overview

In each of the scenarios above the adversary's goal is to tamper with the client code and data under his control. The trusted server's goal is to *detect* any such integrity violations, after which countermeasures (such as severing connections, legal remedies, etc.) can be launched.

Security mechanisms. In this paper we present a system that achieves protection against R-MATE attacks through the extensive use of code diversity and continuous code replacement. In our system, the trusted server continuously and automatically generates diverse variants of client code, pushes these code updates to the untrusted clients, and installs them as the client is running. The intention is to force the client to constantly analyze and re-analyze incoming code variants, thereby overwhelming his analytical abilities, and making it difficult for him to tamper with the continuously changing code without this being detected by the trusted server.

Limitations. Our system specifically targets distributed applications which have frequent client-server communication, since client tampering can only be detected at client-server interaction events. Furthermore, while our use of code diversity can *delay* an attack, it cannot completely *prevent* it. Our goal is therefore the rapid *detection* of attacks; applications which need to completely prevent any tampering of client code, for even the shortest length of time, are not suitable targets for our system. To see this, consider the following timeline in the history of a distributed application running under our system:

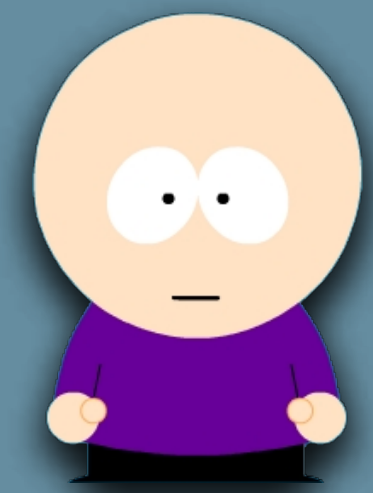


The e_i's are *interaction events*, points in time when clients communicate with servers either to exchange application data or to perform code updates. At time t_1 the client tampers with the code under his control. Until the next interaction event, during interval I_1, the client runs autonomously, and the server cannot detect the attack. At time t_2, after an interval I_2 consisting of a few interaction events, the client's tampering has caused it to display anomalous behavior, perhaps through the use of an outdated communication protocol, and the server detects this. At time t_3, finally, the server issues a response, perhaps by shutting

Really famous authors

To: authors@cs.ux.edu

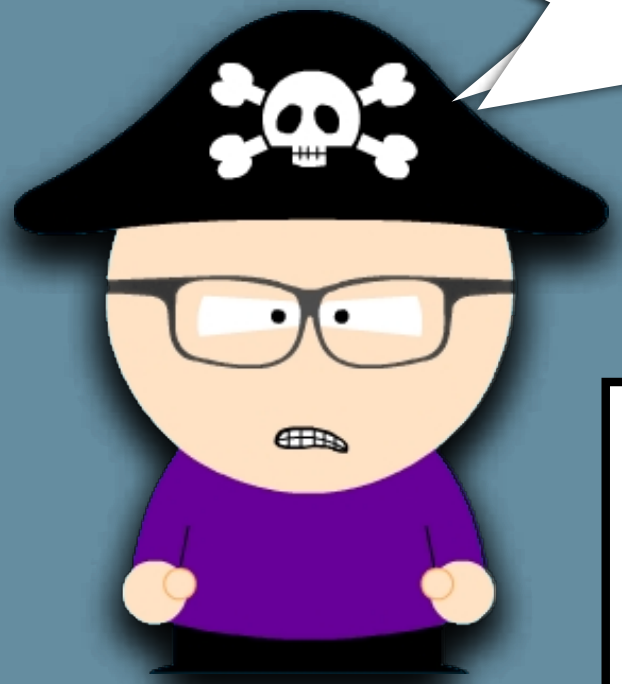
Cool paper! Can you send me the system so I can break it?😀



Technical Report

Conference Paper

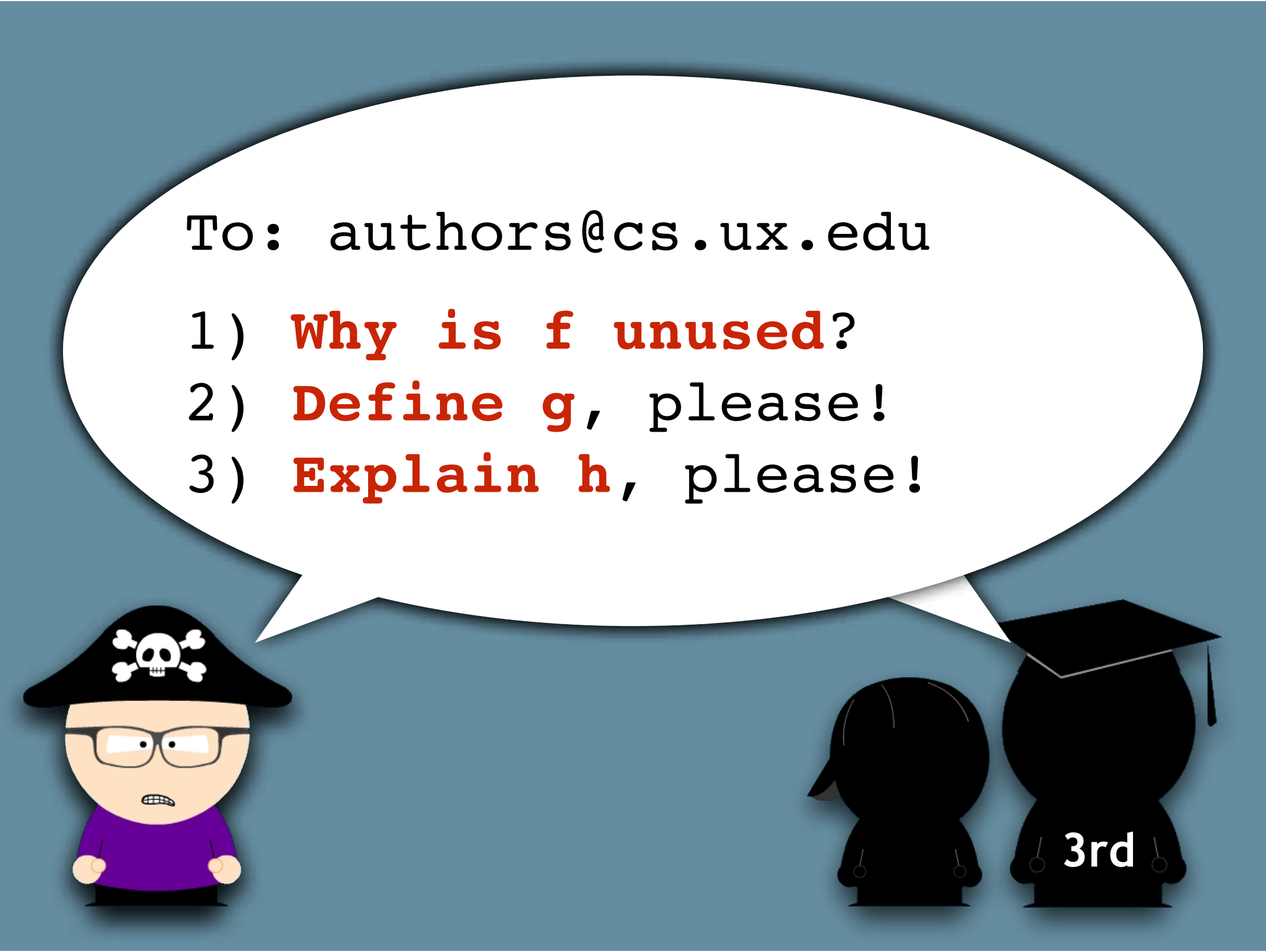
PhD Thesis



f: never used!
g: not defined!
h: makes no sense!

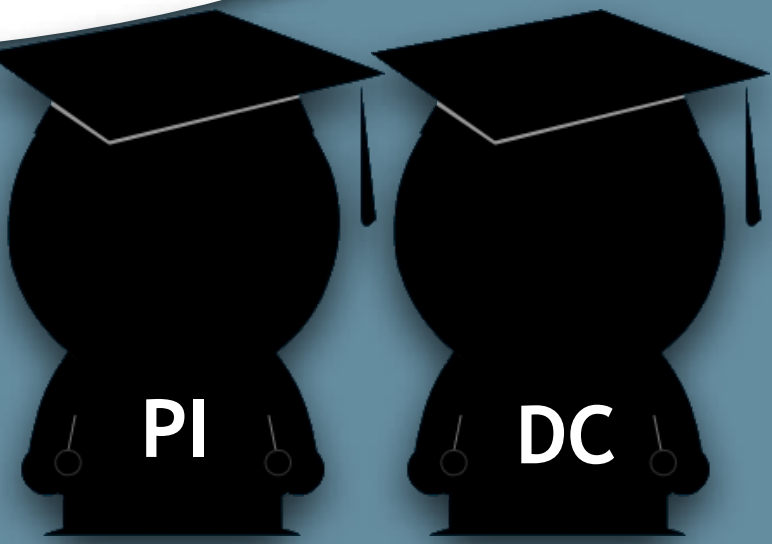
My version of their software!

```
type operator =
    | A
    | B of operand * value * binop
    | C of operand * value * operand * binop
    | D of operand * value * operand * binop
    | E of operand * operand
```


To: authors@cs.ux.edu
1) Why is f unused?
2) Define g, please!
3) Explain h, please!
3rd

To: PI,DC@cs.ux.edu

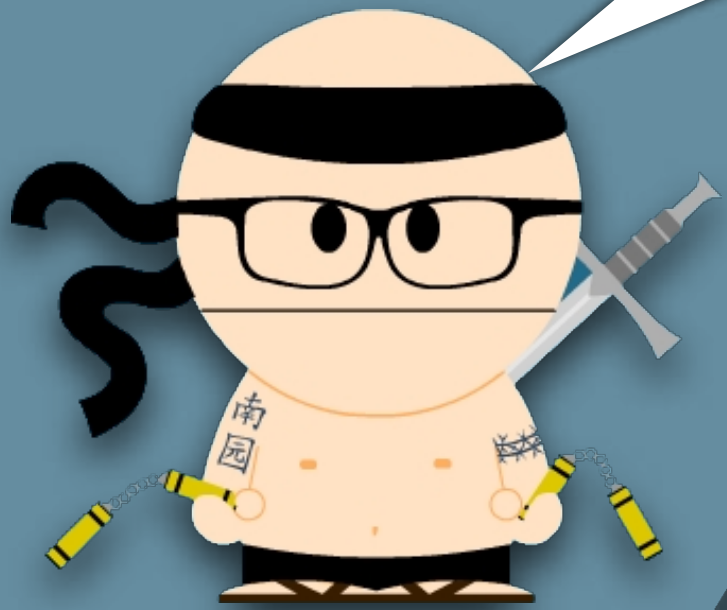
I … **request under the OPEN RECORDS ACT … ALL RESEARCH ARTIFACTS** …



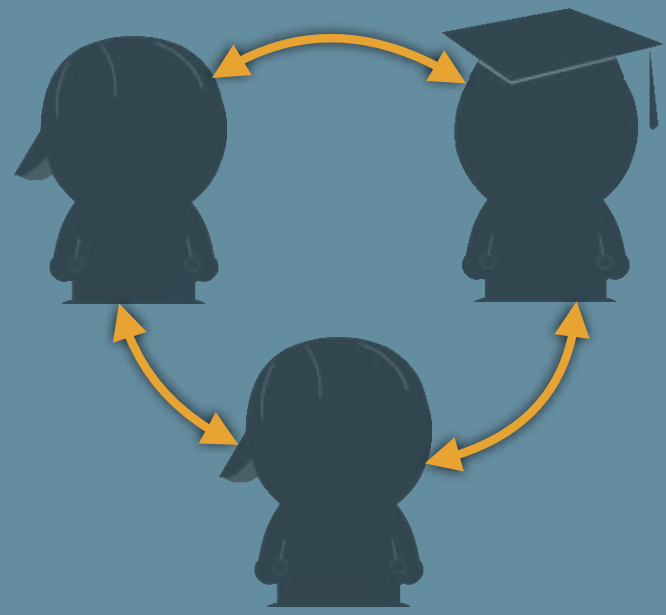
From: legal@cs.ux.edu

… to the extent such records may exist, **they will not be produced pursuant to ORA**.

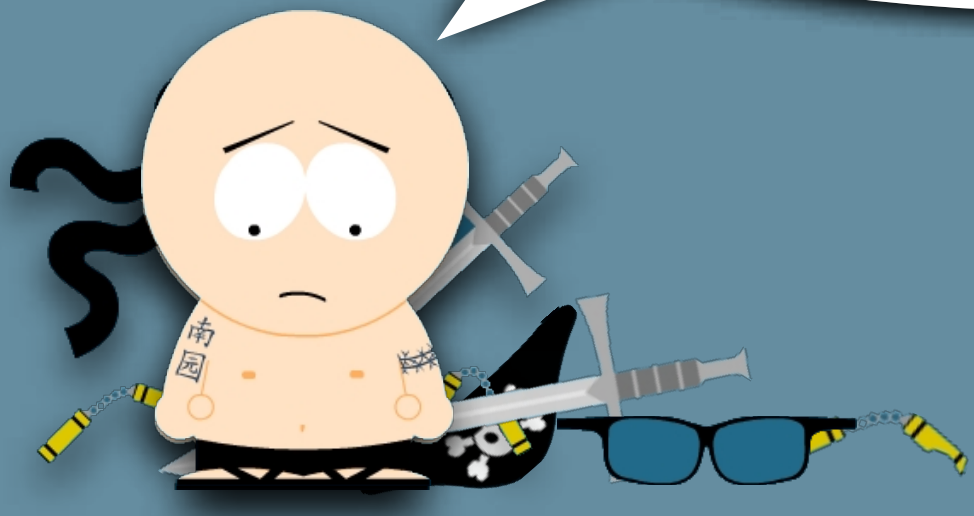
You
can't find it?
Seriously?



PhD
Thesis

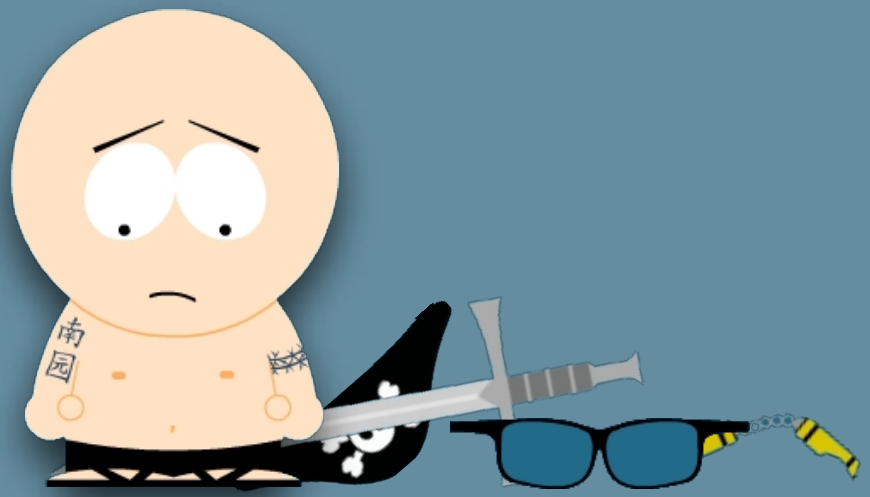


… we estimate a total cost of **$2,263.66** to search for, retrieve, redact and produce such records.



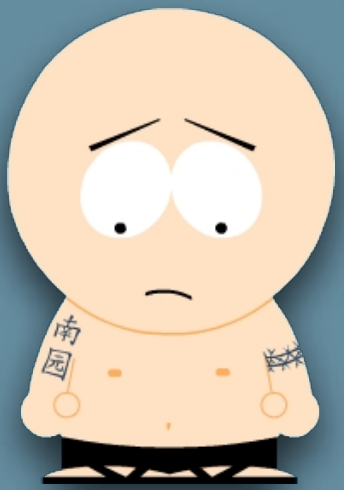
Grant application #: [redacted]

We will also make our data and software available to the research community when appropriate.



Consequences

By not sharing their artifacts, and by (perhaps unintentionally) leaving holes in their publications, the authors have effectively guaranteed that their claims can never be refuted.



1st Law of Artifact Sharing

The probability of getting code out of someone is inversely proportional to the outrageousness of the claims in the paper.

The Deception Study

601 Research Papers

Has artifacts?

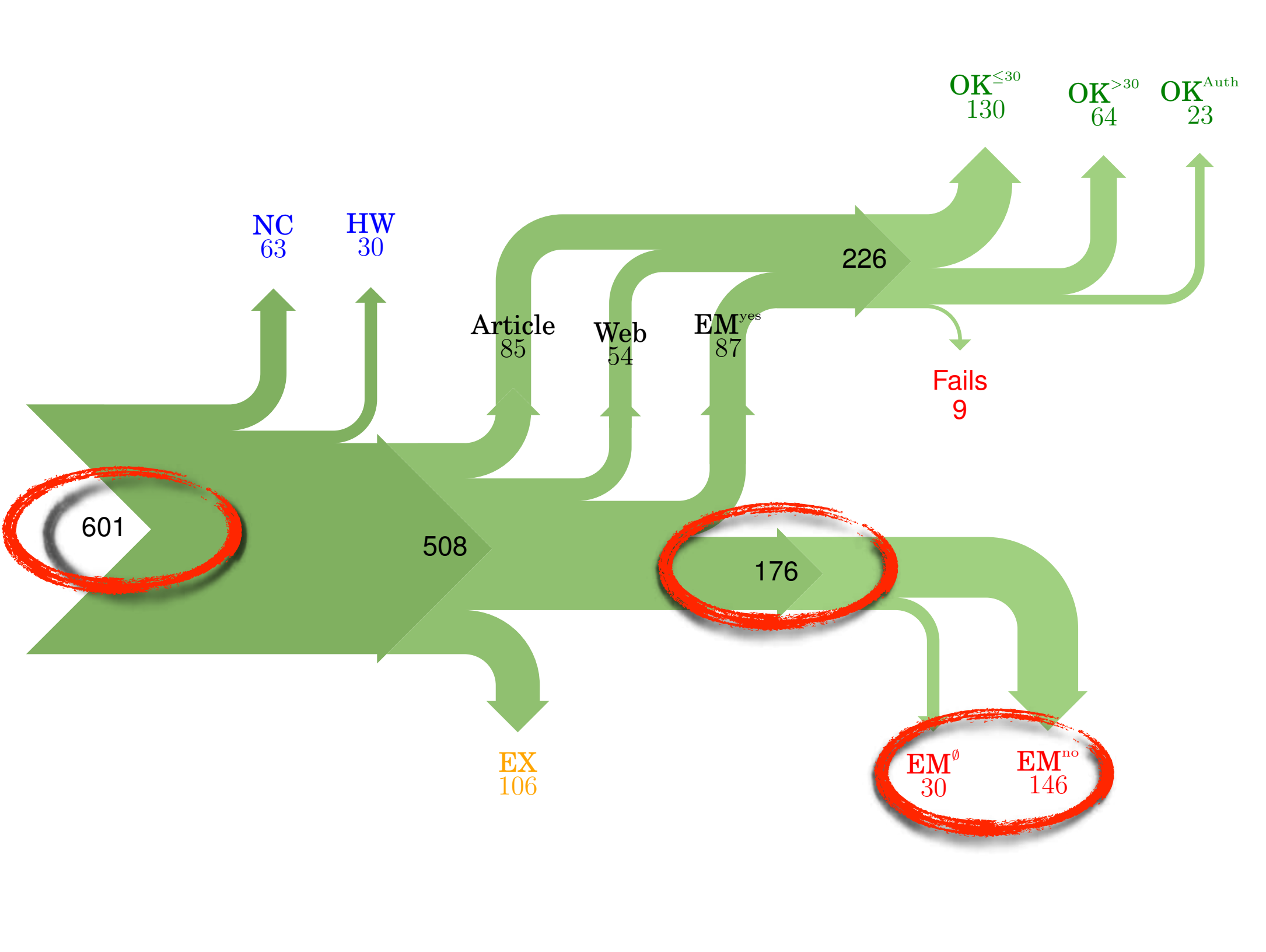
Can we find them?

1. Article?
2. Web?
3. Email?

Do they "work"?

Weakly Repeatable

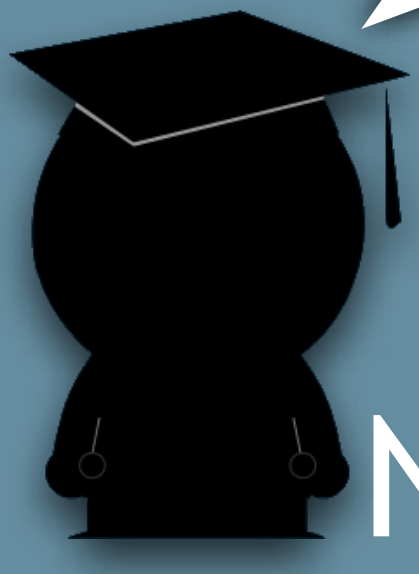
Authors make the artifacts used to create the results in their article available, and they build.


OK≤30
130
OK>30
64
OKAuth
23
NC
63
HW
30
Article
85
Web
54
EMyes
87
226
Fails
9
601
508
176
EX
106
EM∅
30
EMno
146

The good news … I was able to find some code. I am just **hoping** that it … **matches the implementation** we … used for the paper.

Versioning

The code was **never intended to be released** so is not in any shape for general use.



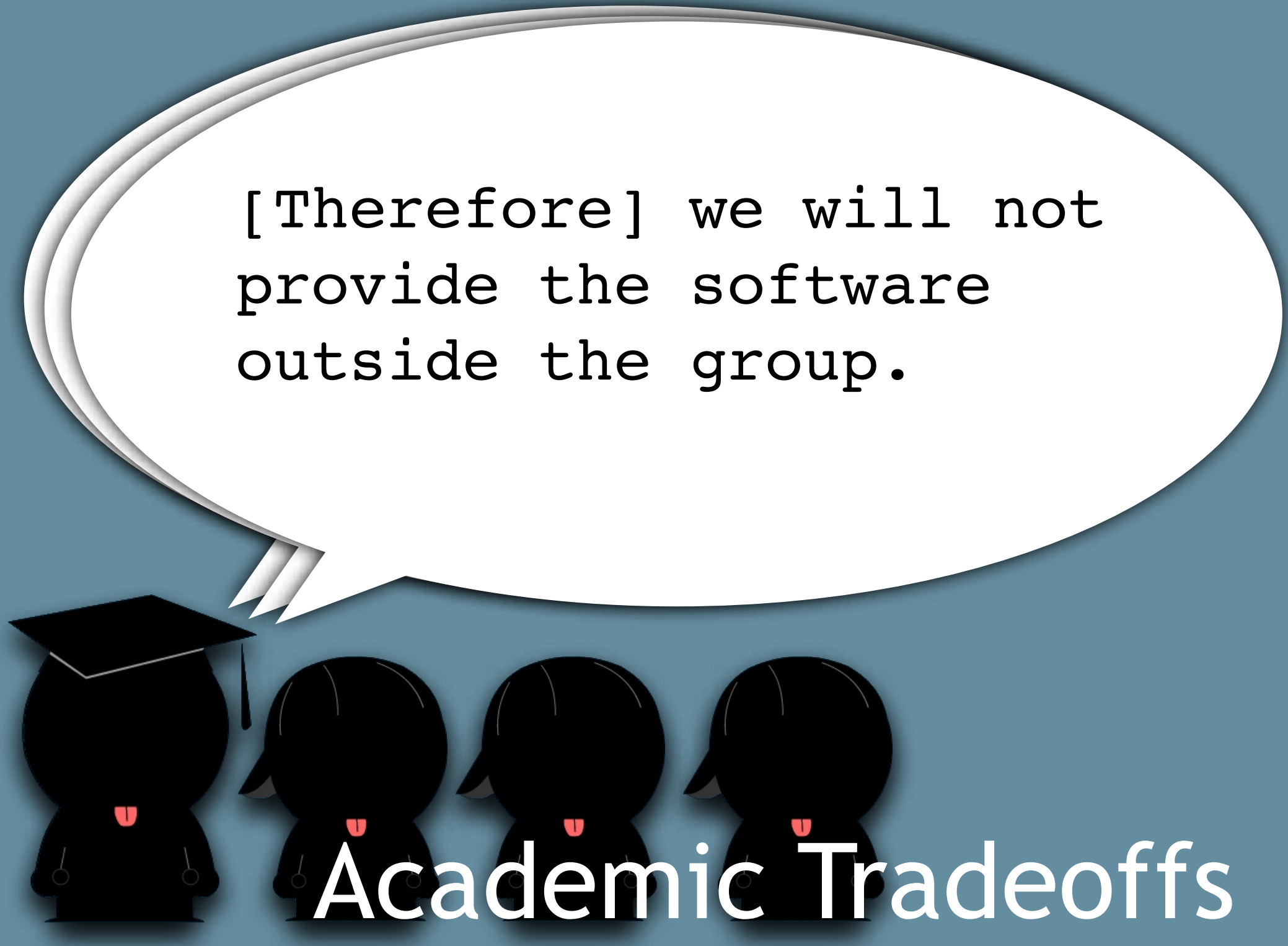
No Intention to Share

[Our] prototype … included many moving pieces that only [student] knew how to operate … **he left**.

Personnel Issues

… the server in which my implementation was stored had a **disk crash** … three disks crashed … Sorry for that.

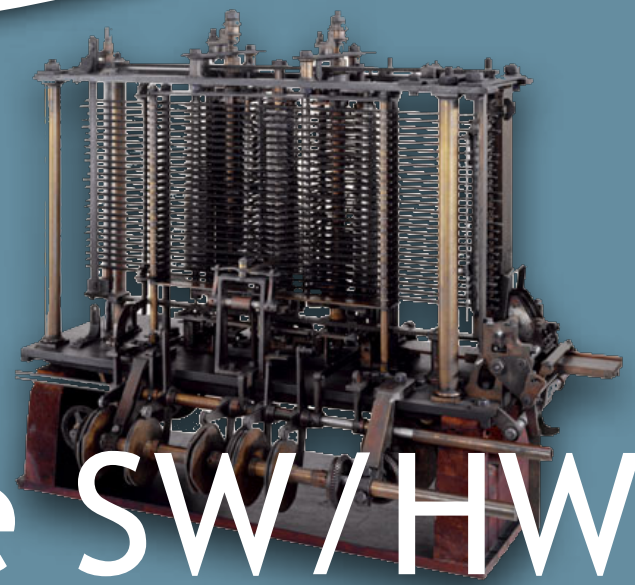
Lost Code


[Therefore] we will not provide the software outside the group.
Academic Tradeoffs

… **we can't share what did for this paper**. … this is not in the academic tradition, but this is a hazard in an industrial lab.

Industrial Lab Tradeoffs

… we have no plans to make the scheduler's source code publicly available. … because [ancient OS] as such **does not exist anymore**



Obsolete SW/HW

… we have an agreement with the [business], and we cannot release the code because of the potential **privacy risks** …

Privacy/Security

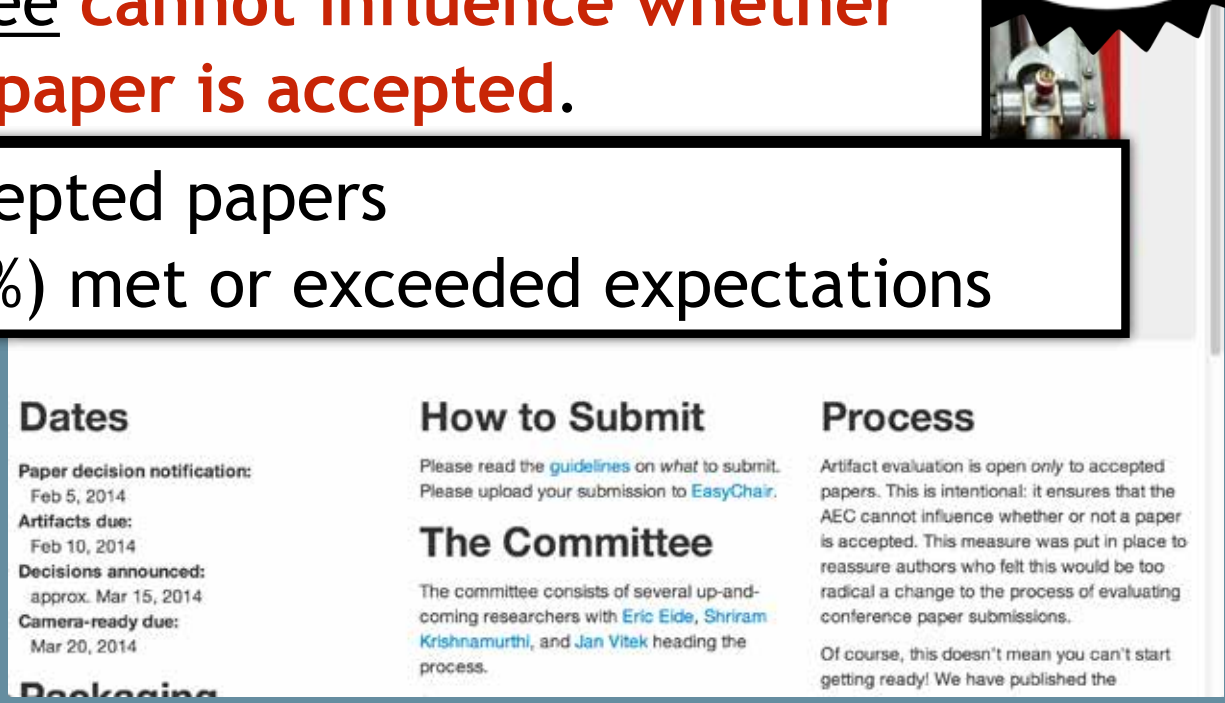
Sharing Proposals

1. Artifact Evaluation
2. Artifact Repositories
3. Funding Agency Audits
4. Sharing Specifications

Artifact Evaluation

Artifact evaluation is open only to accepted papers. This is intentional: it ensures that the Artifact Evaluation Committee **cannot influence whether or not a paper is accepted**.

- 60 accepted papers
- 27 (45%) met or exceeded expectations



Dates

Paper decision notification:
Feb 5, 2014
Artifacts due:
Feb 10, 2014
Decisions announced:
approx. Mar 15, 2014
Camera-ready due:
Mar 20, 2014

How to Submit

Please read the guidelines on *what* to submit. Please upload your submission to EasyChair.

The Committee

The committee consists of several up-and-coming researchers with Eric Eide, Shriram Krishnamurthi, and Jan Vitek heading the process.

Process

Artifact evaluation is open *only* to accepted papers. This is intentional: it ensures that the AEC cannot influence whether or not a paper is accepted. This measure was put in place to reassure authors who felt this would be too radical a change to the process of evaluating conference paper submissions.

Of course, this doesn't mean you can't start getting ready! We have published the

Artifact Repositories

System

- Environment
- Benchmarks

Virtual Machine

- If you build it, they still won't come…

Funding Agency Audits

Agencies should conduct **random audits** to ensure that research artifacts are shared in accordance with what was promised in the grant application

Sharing Specifications

Title

Abstract
Introduction
Keywords

Copyright
Sharing

Provided both in
- submission
- final paper

Clarify which research artifacts will be available

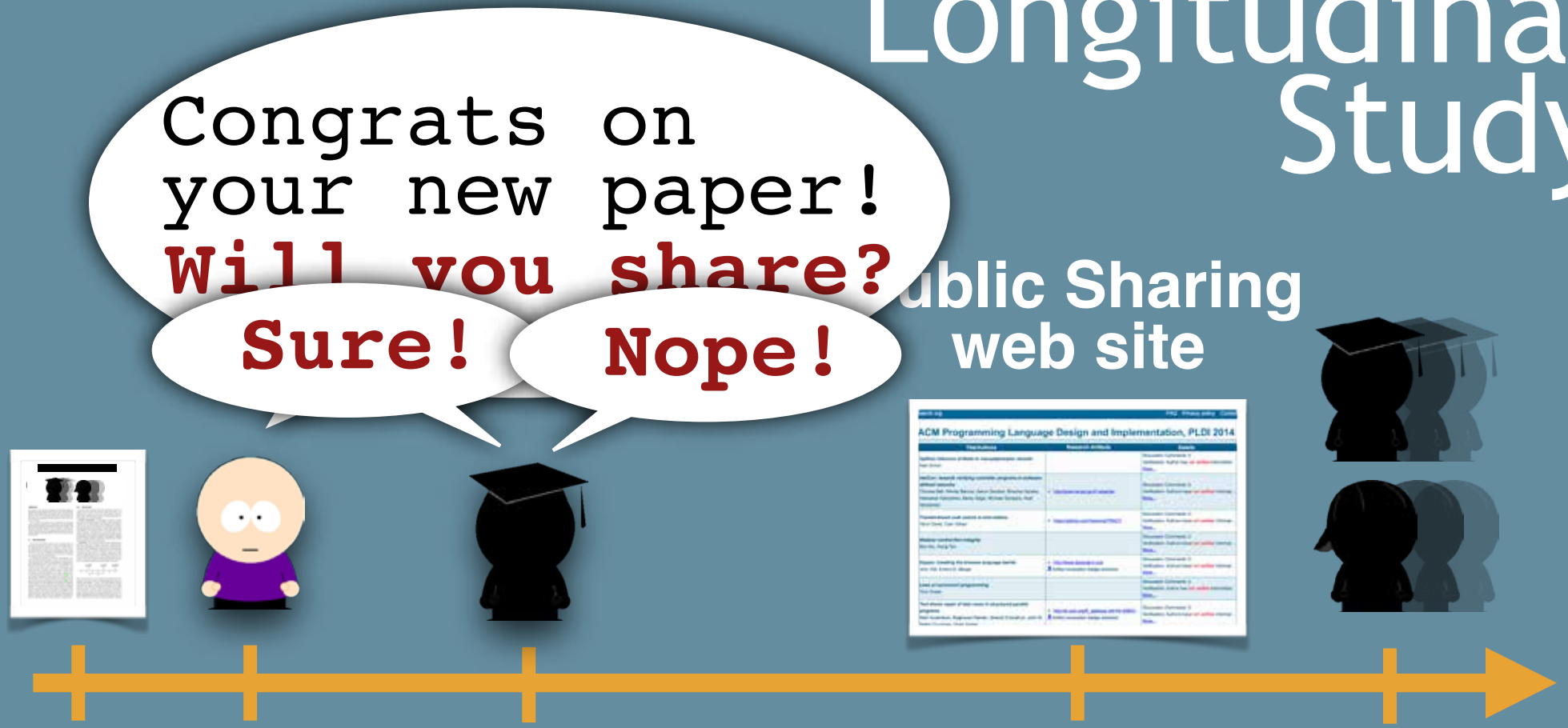
Publishers should require articles to contain a **sharing contract** specifying the level of repeatability to which its authors will commit

Sharing Proposals

5. Artifact Indexing

FindResearch.org

Longitudinal Study



100 conferences,
1000s of authors,
10,000s of papers,
over 5 years!

1. Motivating researchers to share
2. Directory of research artifacts
3. Trending data for funding agencies
4. Data for repeatability research

Sharing, Community
license, discussion!
URL,
contact,
funding...

FindResearch.org FAQ Privacy policy Contact Blog

A Catalog of Research Artifacts for Computer Science

FindResearch.org aim
(e.g., code and data)
essential for repeating

- 79 conferences
- 8721 articles
- 20,675 unique authors
- 28,097 survey emails sent

Se

E.g. Au

Complete results pending while our site is being indexed

Find research artifacts by publication venue

Publication Venue	Year (#artifacts)		
ACM Architectural Support for Programming Languages and Operating Systems, ASPLOS	2016 (20)		
ACM Computer and Communications Security, CCS	2016 (68)	2015 (52)	2014 (22)
ACM Conference on Mobile Systems, Applications, and Services, MobiSys	2016 (8)		
ACM Hypertext and Social Media, HT	2016 (6)	2015 (3)	2014 (9)
ACM Information Retrieval, SIGIR	2016 (62)	2015 (17)	
ACM International Conference on Embedded Software, EMSOFT 2016	2016 (2)		
ACM International Conference on Management of Data, SIGMOD	016 (41)	2015 (27)	
ACM Internet Measurement Conference, IMC	20 (16)		

FindResearch.org Browse FAQ Privacy policy Contact

ACM Principles of Programming Languages, POPL 2017

Title/Authors	Research Artifacts [?]	Details
Serializability for eventual consistency: criterion, analysis, and applications Lucas Brutschy, Dimitar Dimitrov, Peter Müller, Martin T. Vechev	• http://ecracer.inf.ethz.ch/ Artifact evaluation badge awarded	Author Comments: Discussion Comments: 0 Sharing: Research produced artifacts Verification: Authors have verified information More...
Deciding equivalence with sums and the empty type Gabriel S…		Di…ion Comments: 0 Verification: Author has not verified informat…
A seman… Arthur Az… Shin-ya K…		Author Comments: Discussion Comments: 0 Sh…g: Research produced no artifacts Verification: Authors have verified information
Exact Ba… Chung-ch…	…akaru/bl…	Author Comments: Discussion Comments: 0 Sharing: Research produced artifacts Verification: Authors have verified information More...
Intersection type calculi of bounded dimension Andrej Dudenhefner, Jakob Rehof		Discussion Comments: 0 Verification: Authors have not verified infor… More...

- Verified articles: 813 (9%)
- Articles with shared artifacts: 521 (6%)
- Unable to share: 108

This is a great project - keep up the good work, folks!

We'd like to help by providing the information you need for our conference!

I think they might be spying on my computer!

How to Share?

- What to Share?
- How to Package?
- Where to Host?

Your colleagues may need access to an artifact for many different reasons:

- They may want to read the source code to better understand your paper.
- They may want to rerun the experiments you present in your paper to ensure you got things right.
- They may want to build upon and extend your work.
- They may want to compare your results to their own.
- They may want to run different experiments on your code, or run the same experiments you ran but on different data sets.

To ensure *repeatability* provide all the

- sources
- makefiles
- external libraries
- instructions for proof assistants
- data sets
- installation instructions
- scripts to run experiments

that went into producing the results reported on in the final, published, paper.

To ensure *longevity*, future-proof by including all libraries.

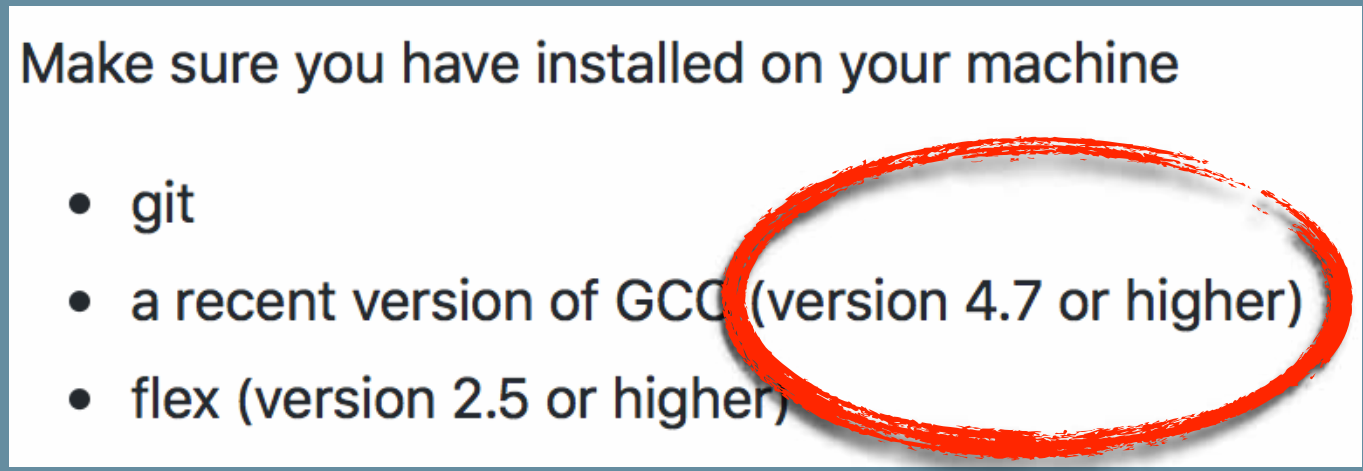
Install the following dependencies:

```
sudo apt-get -y install git g++ cmake libboost-regex-dev
```

Will **`apt-get abclib/1.2.3`** work in 5 years?

To ensure identical binaries are built, precisely document software you can't include:

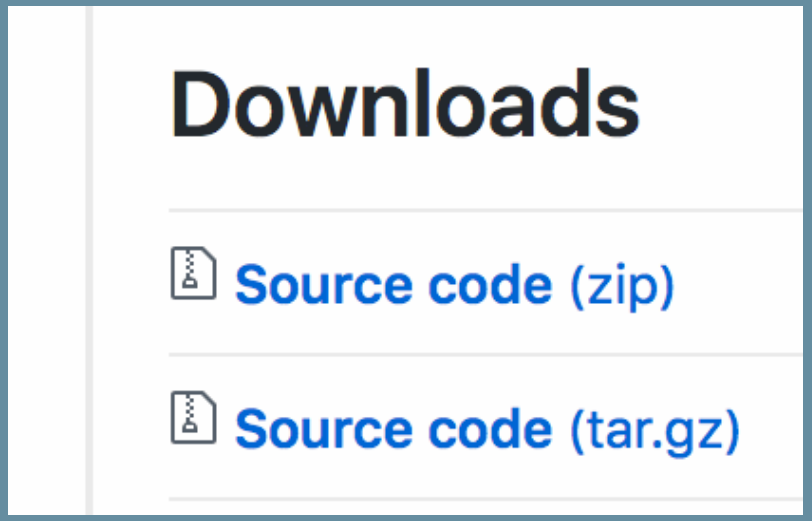
- Name: abclib
- Version: 1.2.3
- Location: `http://abclib.org`



Beware of overloaded package names!

To ensure a *clear mapping* between paper and artifact, provide

- A permanent package (zip-file, virtual machine, container), or
- A "tagged" version of a public repository



2nd Law of Artifact Sharing

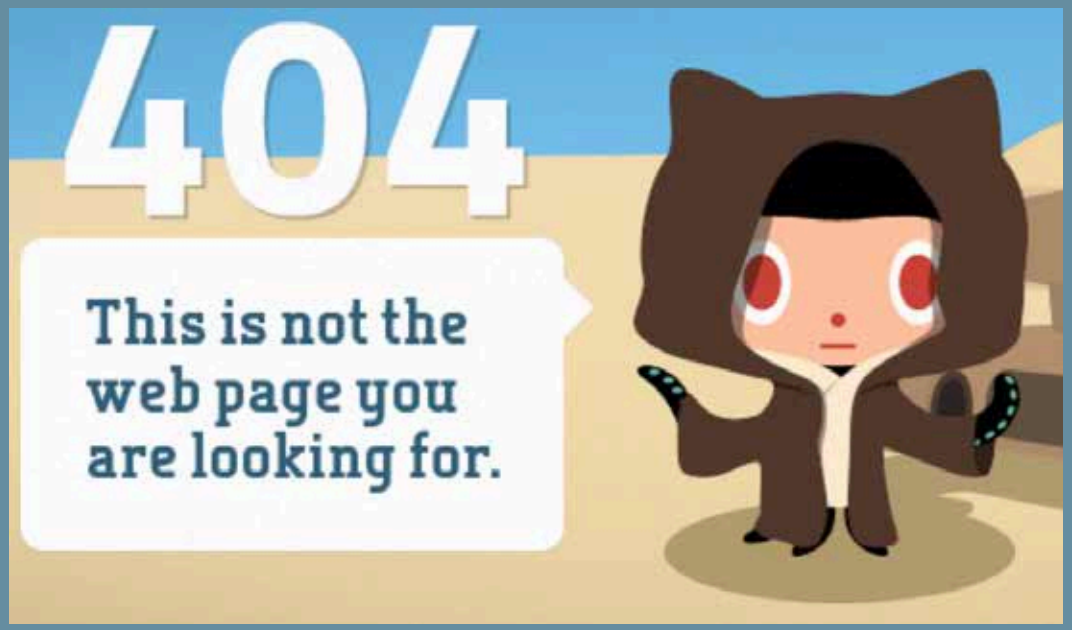
If you can't find which artifact goes with the final published version of a paper, if you don't have every single bit of that artifact, if you don't know the exact environment in which experiments were run, you ain't got nothing.

To ensure availability, host on github, amazon, azure, …

Service Temporarily Unavailable

The server is temporarily unable to service your request due to maintenance downtime or capacity problems. Please try again later.

Apache Server at eventbuilder.geovid.org Port 80



18 verified papers with shared artifacts appear to have broken links (3%)

Some Computer Security Paper

Really Good Computer Science Department
Really Good School

Abstract

We present a new general technique for protecting clients in distributed systems against *Remote Man-at-the-end* (R-MATE) attacks. Such attacks occur in settings where an adversary has physical access to an untrusted client device and can obtain an advantage from tampering with the hardware itself or the software it contains.

In our system, the trusted server overwhelms the untrusted client's analytical abilities by continuously and automatically generating and pushing to him diverse client code variants. The diversity subsystem employs a set of primitive code transformations that provide an ever-changing attack target for the adversary, making tampering difficult without this being detected by the server.

1. Introduction

Man-at-the-end (MATE) attacks occur in settings where an adversary has physical access to a device and compromises it by tampering with its hardware or software. *Remote* man-at-the-end (R-MATE) attacks occur in distributed systems where *untrusted clients* are in frequent communication with *trusted servers* over a network, and malicious user can get an advantage by compromising an untrusted device.

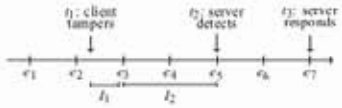
To illustrate the ubiquity of R-MATE vulnerabilities, consider the following four scenarios. First, in the *Advanced Metering Infrastructure* (AMI) for controlling the electrical power grid, networked devices ("*smart meters*") are installed at individual house-holds to allow two-way communication with control servers of the utility company. In an R-MATE attack against the AMI, a malicious consumer tampers with the meter to emulate an imminent blackout, or to trick a control server to send disconnect commands to other customers [7, 21]. Second, massive multiplayer online games are susceptible to R-MATE attacks since a malicious player who tampers with the game client can get an advantage over other players [16]. Third, wireless sensors are often deployed in unsecured environments (such as theaters of war) where they are vulnerable to tampering attempts. A compromised sensor could be coached into supplying the wrong observations to a base station, causing real-world damage. Finally, while electronic health records (EHR) are typically protected by encryption while stored in databases and in transit to doctors' offices, they are vulnerable to R-MATE attack if an individual doctor's client machine is compromised.

1.1 Overview

In each of the scenarios above the adversary's goal is to tamper with the client code and data under his control. The trusted server's goal is to *detect* any such integrity violations, after which countermeasures (such as severing connections, legal remedies, etc.) can be launched.

Security mechanisms. In this paper we present a system that achieves protection against R-MATE attacks through the extensive use of code diversity and continuous code replacement. In our system, the trusted server continuously and automatically generates diverse variants of client code, pushes these code updates to the untrusted clients, and installs them as the client is running. The intention is to force the client to constantly analyze and re-analyze incoming code variants, thereby overwhelming his analytical abilities, and making it difficult for him to tamper with the continuously changing code without this being detected by the trusted server.

Limitations. Our system specifically targets distributed applications which have frequent client-server communication, since client tampering can only be detected at client-server interaction events. Furthermore, while our use of code diversity can *delay* an attack, it cannot completely *prevent* it. Our goal is therefore the rapid *detection* of attacks; applications which need to completely prevent any tampering of client code, for even the shortest length of time, are not suitable targets for our system. To see this, consider the following timeline in the history of a distributed application running under our system:



The e_i's are *interaction events*, points in time when clients communicate with servers either to exchange application data or to perform code updates. At time t_1 the client tampers with the code under his control. Until the next interaction event, during interval I_1, the client runs autonomously, and the server cannot detect the attack. At time t_2, after an interval I_2 consisting of a few interaction events, the client's tampering has caused it to display anomalous behavior, perhaps through the use of an outdated communication protocol, and the server detects this. At time t_3, finally, the server issues a response, perhaps by shutting

To ensure colleagues can ask questions use permanent email addresses

- 2,591 emails bounced (9%)
- 5440 authors without an email address (21%)
- 854 articles without any email addresses (13%)

Pushback

My code is on github, what more do you want???

Sharing≠Repeatability

It's impossibly hard to make a perfect package to share, so why bother?

The perfect is the enemy of the good

If you force sharing for publication, industrial labs can't publish, so let's make sharing optional.

Optional sharing

I really don't have time for this - I need to

- publish the next paper
- ensure my students get jobs
- submit the next grant

The Dean can't read but the Dean can count

3rd Law of Artifact Sharing

The root of the scientific reproducibility problem is sociological, not technological: we do not produce solid artifacts or attempt to replicate the work of our peers because there is little professional glory to be gained from doing so.

Some professor will convince a naive first-year student to use your code as a starting point for their experiments and you will have contributed to that poor student's misery and anguish.

Publishing code is a rather thankless task

Still, it's in the best traditions of science to give our colleagues all the ammunition they need to falsify our results. Thus, **science⇔sharing**.

On the Importance of being Transparent


Questions?